This listing of claims will replace all prior versions, and listings, of claims in the application.

LISTING OF CLAIMS:

1. (Currently Amended) A method of processing a text file in a computer application, comprising the steps:

creating a plurality of templates from samples of the text file, wherein each of the templates has literal fragments of the text file, each of the templates including substitution points that are filled in with application data;

when the format of the text file changes, changing the templates;

providing a macro class to map data from the text file to the computer application;

embedding in one of the templates a pointer to the macro class; and

using said one of the templates as an overlay to parse the text file into segments having data, or as a prototype to generate a segment of an output file;

said using step including the steps of:

i) reaching said pointer in said one of the templates,

ii) when said pointer is reached, using said pointer to invoke said macro class and using said macro class to map data from one of the segments of the text file to the computer application,[[; and]]

iii) said macro class then invoking another one of the templates to further process the text file, and

iv) said macro class handling iterations, conditional logic and preparation of data for said another one of the templates.

2. (Original) A method according to Claim 1, wherein the macro class reads in a segment of the text file and uses the segment to initiate application update processing.

3. (Original) A method according to Claim 1, wherein the macro class derives data from the application and formats it into the text file.

Claim 4 (Cancelled).

5. (Original) A method according to Claim 1, further comprising the step of providing an interface controller to prevent structure clashes by placing text data into appropriate places in a complex object structure as the text file is processed.

6. (Currently Amended) A system which includes a processor and memory for processing a text file in a computer application, ~~comprising~~ the processor being configured for:

~~means for~~ creating a plurality of templates from samples of the text file, wherein each of the templates has literal fragments of the text file, each of the templates including substitution points that are filled in with application data; and when the format of the text file changes, changing the templates;

[[means]] forming a macro class to map data from the text file to the computer application, wherein a pointer to the macro class is embedded in one of the templates;

~~means for~~ using said one of the templates as an overlay to parse the text file into segments having data, or as a prototype to generate a segment of an output file;

said ~~means for~~ processor is configured for performing said using ~~including~~ by

i) ~~means for~~ using said pointer, when said pointer is reached in said one of the templates, to invoke said macros class,

ii) ~~means to use~~ using said macro class to map data from one of the segments of the text file to the computer application[[;]],

iii) ~~means to use~~ using the macro class to invoke another one of the templates to further process the text file, and

iv) using said macro class to handle iterations, conditional logic and preparation of data for said another one of the templates.

7. (Original) A system according to Claim 6, wherein the macro class reads in a segment of the text file and uses the segment to initiate application update processing.

8. (Original) A system according to Claim 6, wherein the macro class derives data from the application and formats it into the text file.

9. (Original) A system according to Claim 6, further comprising an interface controller to prevent structure clashes by placing text data into appropriate places in a complex object structure as the text file is processed.

10. (Currently Amended) A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for processing a text file in a computer application, said method steps comprising:

creating a plurality of templates from samples of the text file, wherein each of the templates has literal fragments of the text file, each of the templates including substitution points that are filled in with application data;

when the format of the text file changes, changing the templates;

providing a macros class to map data from the text file to the computer application;

embedding in one of the templates a pointer to the macro class; and

using said one of the templates as an overlay to parse the text file into segments having data, or as a prototype to generate a segment of an output file;

said using step including the steps of:

i) reaching said pointer in said one of the templates,

ii) when said pointer is reached, using said pointer to invoke said macro class and using said macro class to map data from one of the segments of the text file to the computer application[[;]],

iii) said macro class then invoking another one of the templates to further process the text file, and

iv) said macros class handling iterations, conditional logic and preparation of data for said another one of the templates.

.

11. (Original) A program storage device according to Claim 10, wherein the macro class reads in a segment of the text file and uses the segment to initiate application update processing.

12. (Original) A program storage device according to Claim 10, wherein the macro class derives data from the application and formats it into the text file.

13. (Original) A program storage device according to Claim 10, wherein said method steps further comprise the step of providing an interface controller to prevent structure clashes by placing text data into appropriate places in a complex object structure as the text file is processed.

14. (Previously Presented) A method according to Claim 1, wherein:

the step of using said pointer to invoke said macro class includes the step of passing to said macro class a name for said another template when said macro class is invoked; and

the step of using said macro class includes the step of, said macro class using said name to invoke said another template to further process the text file.

15. (Previously Presented) A system according to Claim 6, wherein:

when the macro class is invoked, a name for said another template is passed to the macro class from said template; and

said macro class uses said name to invoke said another template to further process the text file;

16. (Previously Presented) A program storage device according to Claim 10, wherein

when the macro class is invoked, a name for said another template is passed to the macro class from said template; and

the step of using said macro class includes the steps of, said macro class using said ~~means~~ name to invoke said another template to further process the text file.

17. (Previously Presented) A method according to Claim 5, wherein the step of providing the interface controller includes the steps of:

using the interface controller to set up said complex object structure and to place said text data into said object structure as the text file is processed; and

after the entire text file is processed, using said structure to process updating data into said application.

18. (Previously Presented) A method according to Claim 1, wherein:

the templates support variable substitutions and conditional or iterative generation for output files; and

the method comprising the further steps of:

each macro handling iterations, conditional logic and preparation of data for the next template;

using an interface controller to keep track of the macros needed for a particular process so that the macros do not need to be continually reinvoked;

using literal text within the templates to determine the format of the output stream;

storing the templates for each application as a hashtable that is associated to said each application's definition class;

providing a specialized macro for supporting the navigation of a complex object structure;

including in each template macros that specify points in the input stream from which data are to be taken and what to do with said data;

providing each template with an extraction point to extract data from the input stream, said each template including the name of the field to be extracted and the name of the database table said field belongs to in the current application;

providing each template with a keyword, said keyword being used for data extraction and specifying the column name and the internal class name for the data to be extracted;

including in each template parsing information that specifies, for each field, which internal class the field is assigned to, and the database column name for the field;

using a format class to translate string data from the input field into a suitable format for internal processing; and

providing an interface file to define an interface template for the application, wherein each record in the interface file is matched to a unique template, wherein the type of input record for the template is specified by starting the template name with a table name.

19. (New) A method according to Claim 1, comprising the further steps of:

using literal text within the templates to determine the format of the output steam;

storing the templates for each application as a hashtable that is associated to said each application's definition class; and

providing a specialized macro for supporting the navigation of a complex object structure.